Spark ML

Inteligência na Web e Big Data

Fabricio Olivetti de França e Thiago Ferreira Covões
folivetti@ufabc.edu.br, thiago.covoes@ufabc.edu.br

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

CMCC
Centro de Matemática, Computação e Cognição
Center for Mathematics, Computation and Cognition

Spark ML

ML Pipelines

O Spark possui uma biblioteca específica para *pipeline* de algoritmos de aprendizado de máquina denominado *ML Pipelines*.

Essa biblioteca é inspirada nas bibliotecas Pandas e Scikit-Learn, da linguagem Python, muito utilizada por cientistas de dados.

ML Pipelines

Ela é composta por diversas APIs:

- Dataframe: organiza os dados de entrada de forma estruturada permitindo consultas SQL e diversas manipulações no estilo da biblioteca Pandas
- Transformer: algoritmos que transformam um dataframe em outro, por exemplo, transforma uma base de dados de entrada em uma base de predições.

ML Pipelines

- Estimator: algoritmos que recebem um dataframe de entrada e geram um *transformer*.
- Pipeline: define a cadeia de múltiplos transformers e estimators para gerar um pipeline de processamento.
- Parameter: API compartilhada entre transformes e estimators para definir os parâmetros.

ML Pipelines

Suponha que queremos criar um classificador de documentos de texto. Um *pipeline* natural seria primeiro tokenizar o texto, em seguida gerar atributos numéricos, finalmente aplicar o modelo de classificação e aplicar tal modelo em documentos novos.

ML Pipelines

Um Pipeline é descrito como a sequência de transformações que devem ser feitas para obter o resultado final, seja um modelo ou uma classificação.

ML Pipelines

Vamos primeiro verificar como é feito um pipeline simples, composto apenas de um modelo de regressão logística aplicado em uma base qualquer.

ML Pipelines

Exemplo retirado de https://spark.apache.org/docs/latest/ml-pipeline.html

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.classification import LogisticRegression

training = spark.createDataFrame([
    (1.0, Vectors.dense([0.0, 1.1, 0.1])),
    (0.0, Vectors.dense([2.0, 1.0, -1.0])),
    (0.0, Vectors.dense([2.0, 1.3, 1.0])),
    (1.0, Vectors.dense([0.0, 1.2, -0.5]))], ["label",
↪   "features"])
```

createDataFrame

O método ‘createDataFrame‘ cria um novo DataFrame representado por uma RDD de tuplas em que a primeira posição é o rótulo e a segunda o conjunto de atributos.

O tipo ‘Vectors.dense‘ representa um vetor denso de valores numéricos.

ML Pipelines

```
lr = LogisticRegression(maxIter=10, regParam=0.01)
print("Parâmetros de LogisticRegression:\n" +
 ↪  lr.explainParams() + "\n")

model1 = lr.fit(training)
```

LogisticRegression

O modelo ‘LogisticRegression‘ cria um modelo de regressão logística para classificação binária. Alguns dos parâmetros para esse modelo são:

- maxIter: número de iterações máxima, padrão de 100
- regParam: peso da regularização, padrão em 0.0
- elasticNetParam: peso para uso do elasticNet, padrão em 0.0
- tol: tolerância para decidir se o algoritmo convergiu, padrão em 1e-06
- fitIntercept: se deseja usar o bias, padrão em True
- standardization: se os valores numéricos são centralizados na média, padrão é True

ML Pipelines

```
print("Model 1 was fit using parameters: ")
print(model1.extractParamMap())

paramMap = {lr.maxIter: 20}
paramMap[lr.maxIter] = 30  # Specify 1 Param,
 ↪  overwriting the original maxIter.
paramMap.update({lr.regParam: 0.1, lr.threshold: 0.55})
 ↪  # Specify multiple Params.

paramMap2 = {lr.probabilityCol: "myProbability"}  #
 ↪  Change output column name
paramMapCombined = paramMap.copy()
paramMapCombined.update(paramMap2)
```

Parâmetros

Podemos representar os parâmetros como um dicionário do Python. Dessa forma podemos ter nosso conjunto de parâmetros armazenados em uma variável para serem utilizadas em diversos modelos ou na busca de modelos ótimos.

Além disso é possível mesclar dois dicionários de parâmetros parcialmente preenchidos com o método ‘update‘.

ML Pipelines

```
model2 = lr.fit(training, paramMapCombined)
print("Model 2 was fit using parameters: ")
print(model2.extractParamMap())
```

ML Pipelines

```
test = spark.createDataFrame([
    (1.0, Vectors.dense([-1.0, 1.5, 1.3])),
    (0.0, Vectors.dense([3.0, 2.0, -0.1])),
    (1.0, Vectors.dense([0.0, 2.2, -1.5]))], ["label",
    "features"])

prediction = model2.transform(test)
result = prediction.select("features", "label",
    "myProbability", "prediction") \
    .collect()

for row in result:
    print("features=%s, label=%s -> prob=%s,
    prediction=%s"
          % (row.features, row.label, row.myProbability,
    row.prediction))
```

Predições

As predições geradas pelo método ‘transform‘ são representadas por um DataFrame contendo diferentes colunas com informações do modelo e das predições.

Podemos selecionar as colunas de interesse utilizando o método ‘select‘, e cada linha de resultado é formada por uma tupla nomeada.

ML Pipelines

Vamos seguir um exemplo mais completo de pipeline para processamento e classificação de textos.

ML Pipelines

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer

# Prepare training documents from a list of (id, text,
 ↪  label) tuples.
training = spark.createDataFrame([
    (0, "a b c d e spark", 1.0),
    (1, "b d", 0.0),
    (2, "spark f g h", 1.0),
    (3, "hadoop mapreduce", 0.0)
], ["id", "text", "label"])
```

ML Pipelines

Nesse caso temos, além do texto e do rótulo, um id para cada documento.

ML Pipelines

```
tokenizer = Tokenizer(inputCol="text",
 ↪  outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(),
 ↪  outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Fit the pipeline to training documents.
model = pipeline.fit(training)
```

ML Pipelines

Cada um dos modelos é criado de forma independente e, depois, juntamos todos eles com o objeto ‘Pipeline‘.

Esse objeto possui os mesmos métodos ‘fit‘ e ‘transform‘ para construir o modelo e predizer novos objetos.

ML Pipelines

```
test = spark.createDataFrame([
    (4, "spark i j k"),
    (5, "l m n"),
    (6, "spark hadoop spark"),
    (7, "apache hadoop")
], ["id", "text"])
```

ML Pipelines

```
prediction = model.transform(test)
selected = prediction.select("id", "text",
 ↪  "probability", "prediction")
for row in selected.collect():
    rid, text, prob, prediction = row
    print("(%d, %s) --> prob=%s, prediction=%f" % (rid,
 ↪  text, str(prob), prediction))
```